4-5-2021

Lecture 10: Preconditioned Iterative Methods for Linear Systems

Edmond Chow
Georgia Institute of Technology

Preconditioned Iterative Methods for Linear Systems

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology, USA
https://cse.gatech.edu

46th University of Arkansas Spring Lecture Series
April 5, 2021

Iterative methods for solving linear systems

Given nonsingular matrix A and vector b,

$$\text{solve } Ax = b$$

Two major classes of solution methods:

1. Direct methods
   e.g., $A = LU$, $x = U^{-1}(L^{-1}b)$

2. Iterative methods

```
    $x^{(0)}$ = initial approximation
    **for** $j = 0, 1, \ldots$ until convergence **do**
        $x^{(j+1)} = x^{(j)} + \boxed{\phantom{xxxxxxxx}}$
    **end for**
```

   where $\boxed{}$ generally involves a matrix-vector multiplication with A

Stationary iterative methods (linear fixed-point iteration)

Choose a splitting $A = M - N$

- M nonsingular and may approximate A
- "easy" to solve with M

$$\begin{aligned} (M-N)x &= b \\ x &= (M^{-1}N)x + M^{-1}b \end{aligned}$$

Stationary iterative method:

$$x^{(j+1)} = x^{(j)} + \boxed{M^{-1}(b - Ax^{(j)})}$$

Define the residual $r^{(j)} = b - Ax^{(j)}$ and the error $e^{(j)} = x - x^{(j)}$

$$e^{(j+1)} = \underbrace{(I - M^{-1}A)}_{\text{iteration matrix}} e^{(j)}$$

Semi-iterative methods

Example (Chebyshev method):

$$x^{(j+1)} = x^{(j)} + \sum_{i=0}^{j} \gamma_{j,i} r^{(i)}$$

$$\begin{aligned} x^{(1)} &= x^{(0)} + \gamma_{0,0} r^{(0)} \\ x^{(2)} &= x^{(1)} + \gamma_{1,1} r^{(1)} + \gamma_{1,0} r^{(0)} \\ x^{(3)} &= x^{(2)} + \gamma_{2,2} r^{(2)} + \gamma_{2,1} r^{(1)} + \gamma_{2,0} r^{(0)} \\ &\vdots \end{aligned}$$

$$e^{(j)} = p_j(A) e^{(0)}, \quad p_j(0) = 1$$

$$\|e^{(j)}\| \leq \|p_j(A)\| \|e^{(0)}\|$$

Semi-iterative methods

If A is SPD,

$$\begin{aligned} \min \|p_j(A)\|_2 &= \min_{p_j} \max_{\lambda \in \text{spec}(A)} |p_j(\lambda)| \\ &\approx \min_{p_j} \max_{\lambda_{\min} \le s \le \lambda_{\max}} |p_j(s)| \end{aligned}$$

The approximate solution $p_j(s)$ is a shifted and scaled Chebyshev polynomial, but we need to estimate $\lambda_{\min}$ and $\lambda_{\max}$.

Krylov subspace methods

Find $x^{(m)} \in x^{(0)} + K_m(A, r^{(0)})$, $\quad K_m(A, r^{(0)}) = \text{span}\{r^{(0)}, Ar^{(0)}, A^2 r^{(0)}, \dots, A^{m-1} r^{(0)}\}$

$$\begin{aligned}
x^{(m)} &= x^{(0)} + \alpha_0 r^{(0)} + \alpha_1 A r^{(0)} + \alpha_2 A^2 r^{(0)} + \cdots + \alpha_{m-1} A^{(m-1)} r^{(0)} \\
&= x^{(0)} + q_{m-1}(A) r^{(0)} \\
e^{(m)} &= e^{(0)} - q_{m-1}(A) A e^{(0)} \\
e^{(m)} &= \underbrace{(I - A q_{m-1}(A))}_{p_m(A) \text{ with } p_m(0)=1} e^{(0)}
\end{aligned}$$

$x^{(m)}$ is chosen from $x^{(0)} + K_m(A, r^{(0)})$ to (exactly or approximately) minimize $\|r^{(m)}\|_2$ or $\|e^{(m)}\|_A$ (if A is SPD).

Krylov subspace methods

Formally, to solve $\min \|e^{(m)}\|_A = \min \|p_m(A)e^{(0)}\|_A$:

let the columns of V_m be a basis for $K_m(A, r^{(0)})$ (Arnoldi/Lanczos algorithm), then

$$x^{(m)} = x^{(0)} + \underbrace{V_m(V_m^T A V_m)^{-1} V_m^T}_{\text{approx } A^{-1}} r^{(0)}$$

Note analogy to 1 cycle of a 2-level method:

$$x^{(1)} = x^{(0)} + P(P^T A P)^{-1} P^T r^{(0)}$$

where P is the (sparse) interpolation operator and P^T is the restriction operator.

- No estimation of the spectrum of A was needed to find $p_m(A)$
- However, inner-product operations are now required (Arnoldi/Lanczos)

Preconditioning

Goal: improve the convergence rate of an iterative method with the transformation

$$M^{-1}Ax = M^{-1}b \quad \text{(left preconditioning)}$$

For SPD matrix A and SPD M,

- reduce $\lambda_{\max}(M^{-1}A)/\lambda_{\min}(M^{-1}A)$
- cluster the eigenvalues of $M^{-1}A$
- move the spectrum farther from zero

Preconditioning: another goal

Given a matrix A_μ (that depends on a parameter μ),
find M_μ such that there exist constants c_1, c_2:

$$c_1 x^T M x \leq x^T A x \leq c_2 x^T M x, \quad \text{for all } x \neq 0,$$

i.e., the condition number of $M^{-1}A$ is bounded above by c_2/c_1 *independently* of μ.

Preconditioned conjugate gradient method

```
$x^{(0)} =$ initial approximation
$r^{(0)} = b - Ax^{(0)}$
$p^{(0)} = M^{-1} r^{(0)}$
$z^{(0)} = M^{-1} r^{(0)}$
**for** $j = 0, 1, \dots$ until convergence **do**
    $\alpha^{(j)} = (r^{(j)}, z^{(j)})/(p^{(j)}, Ap^{(j)})$
    $x^{(j+1)} = x^{(j)} + \alpha^{(j)} p^{(j)}$
    $r^{(j+1)} = r^{(j)} - \alpha^{(j)} Ap^{(j)}$
    $z^{(j+1)} = M^{-1} r^{(j+1)}$
    $\beta^{(j)} = (r^{(j+1)}, z^{(j+1)})/(r^{(j)}, z^{(j)})$
    $p^{(j+1)} = z^{(j+1)} + \beta^{(j)} p^{(j)}$
**end for**
```

Main cost of algorithm:

- one matrix-vector multiplication and one preconditioning operation per iteration
- two dot products per iteration

Parallelization

Matrix-vector multiplication ("loosely" synchronous)

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1P} \\ A_{21} & A_{22} & \cdots & A_{2P} \\ \vdots & & \ddots & \vdots \\ A_{P1} & A_{P2} & \cdots & A_{PP} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_P \end{bmatrix}$$

Inner product when result must be distributed (synchronous)

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_P \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_P \end{bmatrix}$$

Load imbalance may cause processors to be idle.

Avoiding/reducing the cost of parallel inner products

- Pipelined algorithms: in CG, it is possible to compute the two inner products together, and overlap these inner products with the matrix-vector multiplication (Ghysels and Vanroose, 2014)
- Preconditioning, especially polynomial preconditioning:

  $$M^{-1}r = p(A)r$$

  (for recent work, see Embree, Loe, and Morgan, 2021)
- Hybrid methods:

  Phase 1 with inner products: Estimate the eigenvalues of A
  Phase 2 with no inner products: Construct and apply a polynomial method $p(A)$ using these estimates

  (for a review, see Nachtigal, Reichel, and Trefethen, 1992)

Two topics in more depth

- Asynchronous iterative methods
- Fine-grained parallel incomplete factorization preconditioning

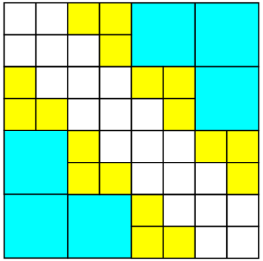
Preconditioned iterative methods for large, dense kernel matrices

Given n points x_i, $i = 1, \dots, n$, define the $n \times n$ dense SPD kernel matrix A with entries

$$a_{ij} = \kappa(x_i, x_j)$$

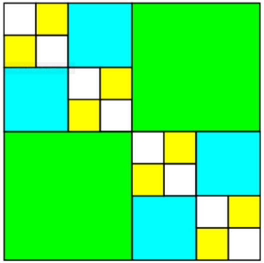
To solve iteratively with A, we desire matrix-vector multiplications by A in $O(n)$ time as well as a SPD preconditioner M for A.

Represent A in $\mathcal{H}^2$ format:



Inexpensive to construct.

Construct M in HSS format:

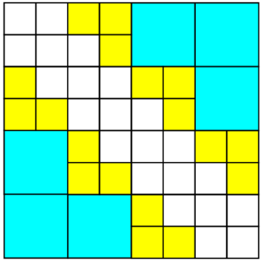


Expensive to construct.

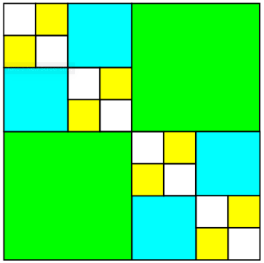
Use the $\mathcal{H}^2$ matrix to help rapidly construct the HSS matrix.

Preconditioned iterative methods for large, dense kernel matrices

Represent A in $\mathcal{H}^2$ format:



Construct M in HSS format:



X. Xing, H. Huang, and E. Chow, Efficient construction of an HSS preconditioner for symmetric positive definite H2 matrices, SIAM Journal on Matrix Analysis and Applications, to appear (2021).

H. Huang, X. Xing, and E. Chow, H2Pack: High-Performance H2 Matrix Package for Kernel Matrices Using the Proxy Point Method, ACM Transactions on Mathematical Software, 47, No. 1, Article 3 (2020).

https://github.com/scalable-matrix/H2Pack/

Asynchronous iterative methods

Fixed-point iteration for solving $x = G(x)$

$$x^{(j+1)} = G(x^{(j)}), \quad j = 0, 1, 2, \ldots$$

Written explicitly:

$$\begin{aligned}
x_1 &\leftarrow g_1(x_1, x_2, x_3, x_4, \cdots, x_n) \\
x_2 &\leftarrow g_2(x_1, x_2, x_3, x_4, \cdots, x_n) \\
x_3 &\leftarrow g_3(x_1, x_2, x_3, x_4, \cdots, x_n) \\
x_4 &\leftarrow g_4(x_1, x_2, x_3, x_4, \cdots, x_n) \\
&\vdots \\
x_n &\leftarrow g_n(x_1, x_2, x_3, x_4, \cdots, x_n)
\end{aligned}$$

Two factors make asynchronous iterations different from synchronous iterations:

- Not all updates are performed at the same time instant
- Updates may use stale information (communication delays in reading or writing)

Mathematical model of asynchronous iterations

Bertsekas and Tsitsiklis 1989. (See also Chazan-Miranker 1969, Baudet 1978.)
Let $x_i^{(j)}$ denote the ith component of x at time *instant* j.

$$x_i^{(j)} = \begin{cases} x_i^{(j-1)}, & \text{if } i \notin J_j \\ g_i(x_1^{(s_1^i(j))}, x_2^{(s_2^i(j))}, \dots, x_n^{(s_n^i(j))}), & \text{if } i \in J_j \end{cases}$$

where J_j is the set of indices updated at instant j, and $s_k^i(j)$ is the instant that x_k is read when computing g_i at instant j (and account for read/write communication delays).

Assumptions:

1. $s_k^i(j) < j$, i.e., at instant j, a process cannot read other values computed at instant j or in the future of instant j
2. Cannot have a sequence $s_k^i(j)$ over time such that iterations only use old values, i.e., newer values must eventually be used
3. No component i is abandoned forever, i.e., no process stops updating

The model is very general and includes synchronous iterations as special cases.

Mathematical model of asynchronous iterations

Bertsekas and Tsitsiklis 1989. (See also Chazan-Miranker 1969, Baudet 1978.)
Let $x_i^{(j)}$ denote the ith component of x at time *instant* j,

$$x_i^{(j)} = \begin{cases} x_i^{(j-1)}, & \text{if } i \notin J_j \\ g_i(x_1^{(s_1^i(j))}, x_2^{(s_2^i(j))}, \ldots, x_n^{(s_n^i(j))}), & \text{if } i \in J_j \end{cases}$$

where J_j is the set of indices updated at instant j, and $s_k^i(j)$ is the instant that x_k is read when computing g_i at instant j (and account for read/write communication delays).

- It is reasonable (but not necessary) to assume $s_k^i(j_1) \le s_k^i(j_2)$ if $j_1 < j_2$, i.e., once a value of x_k is read, an older version of x_k is not read.
- It could be reasonable to assume that, for a given k and j, all $s_k^i(j)$ are equal for different i, i.e., at instant j, all computations are performed with the same values of x.
- When x_i depends on x_i itself (not in the case of Jacobi), $x_i^{(j)}$ might not be computed using the latest value $x_i^{(j-1)}$. This handles the case that the computation of x_i is assigned to different processes at different times.

Convergence theorem for linear case

To solve the nonsingular system $Ax = b$, rewrite the equation in the form $x = G(x)$ as

$$\begin{aligned}(M - N)x &= b \\ x &= (M^{-1}N)x + M^{-1}b\end{aligned}$$

Define the iteration matrix $T = M^{-1}N$. The corresponding synchronous iterative method converges for any initial guess *if and only if* $\rho(T) < 1$.

Convergence theorem for linear case

To solve the nonsingular system $Ax = b$, rewrite the equation in the form $x = G(x)$ as

$$\begin{aligned} (M - N)x &= b \\ x &= (M^{-1}N)x + M^{-1}b \end{aligned}$$

Define the iteration matrix $T = M^{-1}N$. The corresponding synchronous iterative method converges for any initial guess *if and only if* $\rho(T) < 1$.

The corresponding asynchronous iterative method converges for any initial guess *if and only if*

$$\rho(|T|) < 1.$$

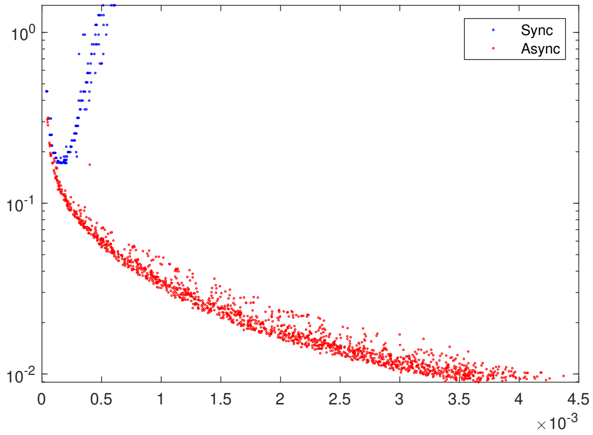
Note: $\rho(T) \leq \rho(|T|)$.

If $\rho(|T|) \geq 1$, then there exists an initial guess $x^{(0)}$ and a sequence of asynchronous iterations that does not converge to the fixed point.

Synchronous and asynchronous Jacobi convergence with 20 threads

2D isotropic diffusion PDE with unstructured FEM discretization, $n = 3081$

$\rho(T) = \rho(|T|) > 1$



The convergence theory gives an overly negative view of asynchronous iterative methods

For synchronous methods, $\rho(T) < 1$ reliably predicts that the iterations will converge for any initial guess.

If $\rho(T) \geq 1$, the iterations "converge" if the initial error does not lie in the subspace spanned by the eigenvectors of T corresponding to eigenvalues ≥ 1.

The convergence theory gives an overly negative view of asynchronous iterative methods

For synchronous methods, $\rho(T) < 1$ reliably predicts that the iterations will converge for any initial guess.

If $\rho(T) \geq 1$, the iterations "converge" if the initial error does not lie in the subspace spanned by the eigenvectors of T corresponding to eigenvalues ≥ 1.

For asynchronous methods, $\rho(|T|) < 1$ guarantees the iterations will converge.

But, if $\rho(|T|) \geq 1$, the iterations may converge anyway (seems more likely if $\rho(|T|)$ is not too large).

Simulating asynchronous iterations

Asynchronous iterations vs. Synchronous iterations

- Differences in convergence (e.g., use of stale values)
- Differences in computational efficiency (e.g., synchronizations, data copies)

Simulating asynchronous iterations lets us study convergence independently of computational efficiency.

Simulation parameters:

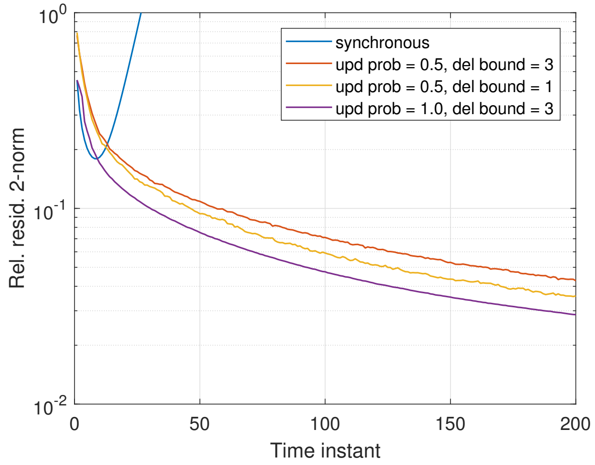
- Update probability: at each time instant a variable is updated with a given probability
- Delay bound: when a variable is updated, the data is read from k past time instants, up to the delay bound ($k = 1$ is most recent data)

J. Wolfson-Pou and E. Chow, "Convergence models and surprising results for the asynchronous Jacobi method," 32nd IEEE Int. Parallel & Distributed Processing Symp., 940-949 (2018).

J. Wolfson-Pou and E. Chow, "Modeling the asynchronous Jacobi method without communication delays," *J. Parallel & Distributed Computing*, 128, 84-98 (2019).

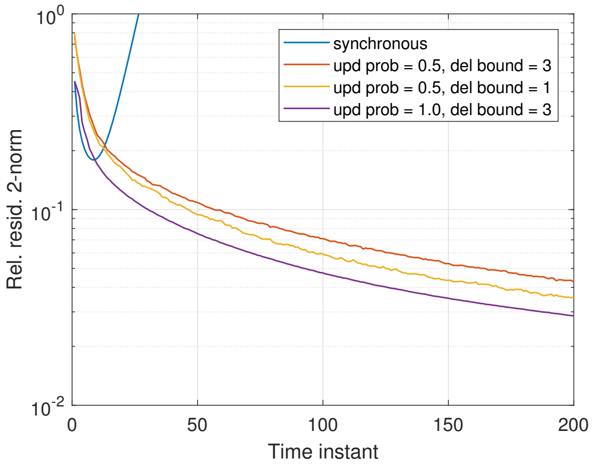
Simulation of asynchronous Jacobi convergence

2D isotropic diffusion PDE with unstructured FEM discretization, $n = 3081$
$\rho(T) = \rho(|T|) > 1$



Two mysteries

1. Why does asynchronous Jacobi converge when some updates are not performed at every step?
2. Why does asynchronous Jacobi converge when some updates use stale values?



Weighted Jacobi

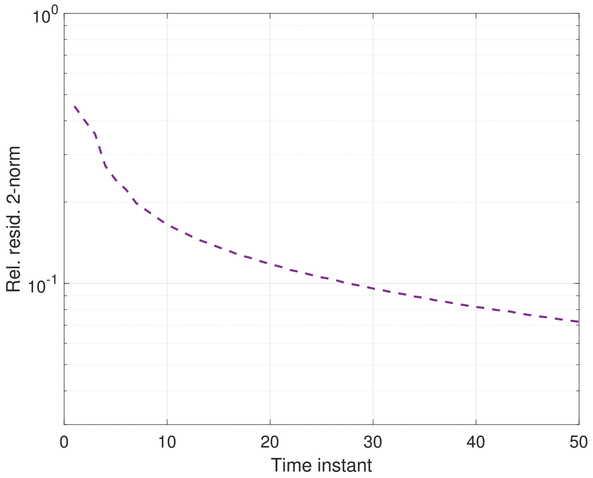
To understand the effect of using stale values, one can think of comparing to weighted Jacobi iterations:

$$
\begin{aligned}
x_i^{(j)} &= (1-\omega)x_i^{(j-1)} + \omega\, g_i(x^{(j-1)}) \\
&= x_i^{(j-1)} + \omega \frac{1}{a_{ii}}(b_i - \sum_k a_{ik} x_k^{(j-1)})
\end{aligned}
$$

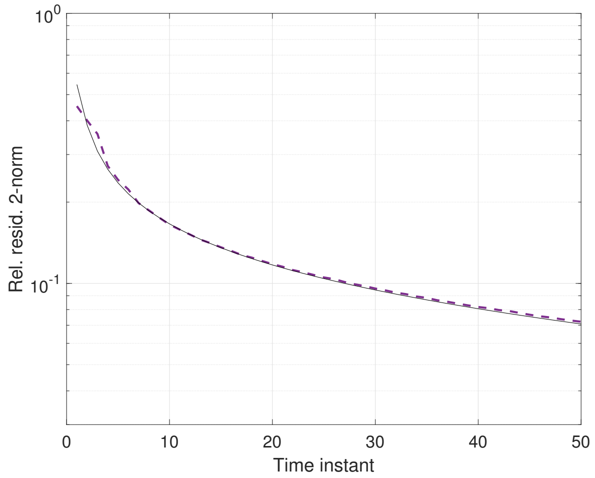
If $\omega < 1$, the new iterate does not get the full correction compared to standard Jacobi.

For SPD systems, if Jacobi does not converge, weighted Jacobi can obtain convergence.

Asynchronous Jacobi with delay bound 3

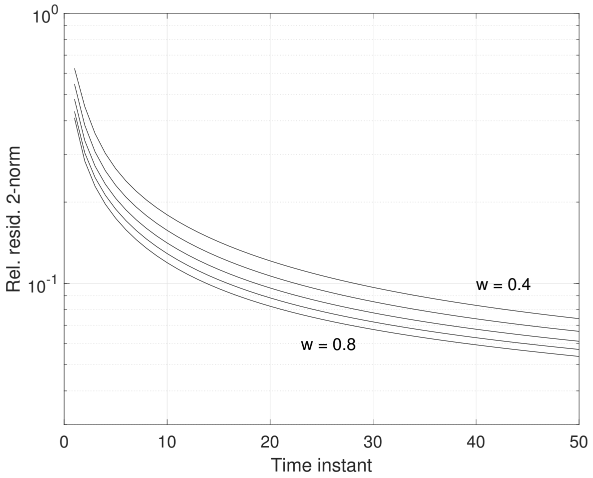


Asynchronous Jacobi with delay bound 3 and w-Jacobi(0.5)

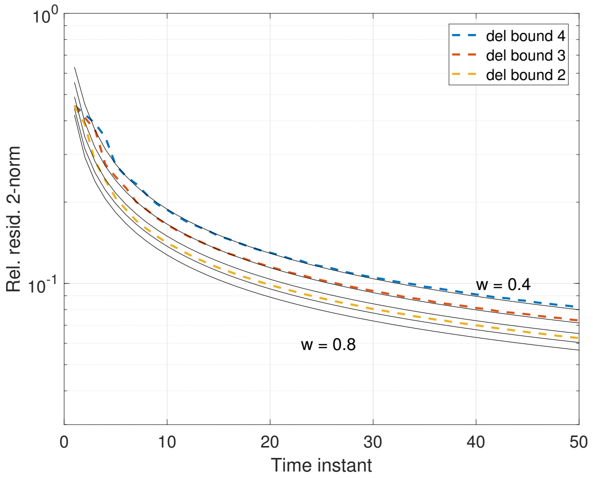


Iterates are not the same, but the residual norms match well.
(Discrepancies are smaller than those for different right-hand sides.)

Weighted Jacobi

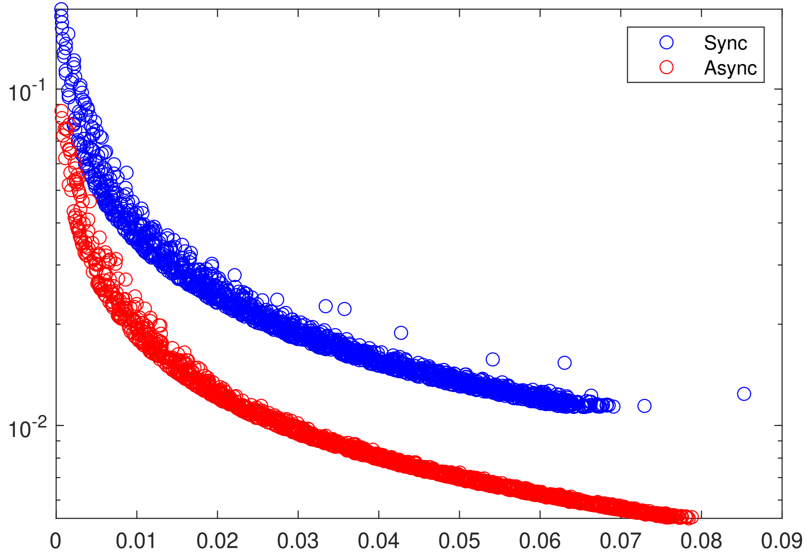


Weighted Jacobi seems to model asynchronous Jacobi using stale values

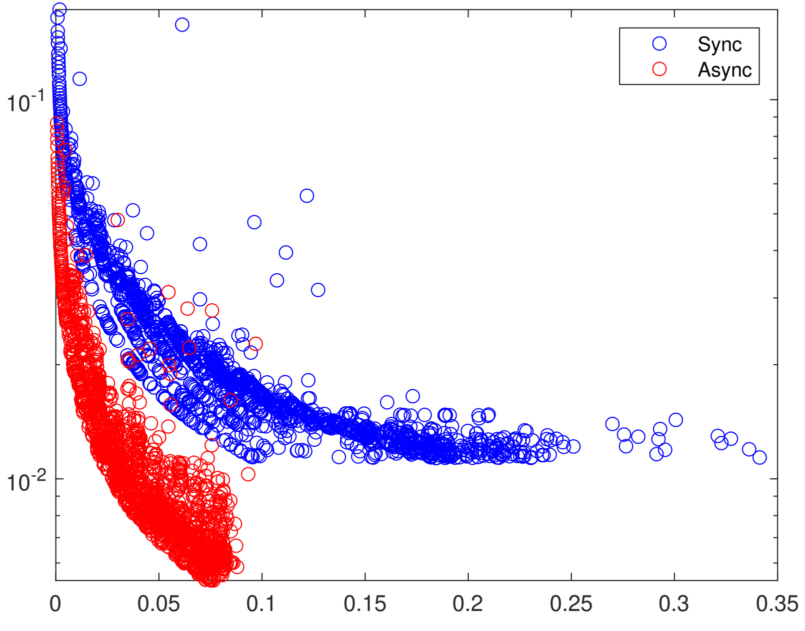


Synchronous and asynchronous Jacobi

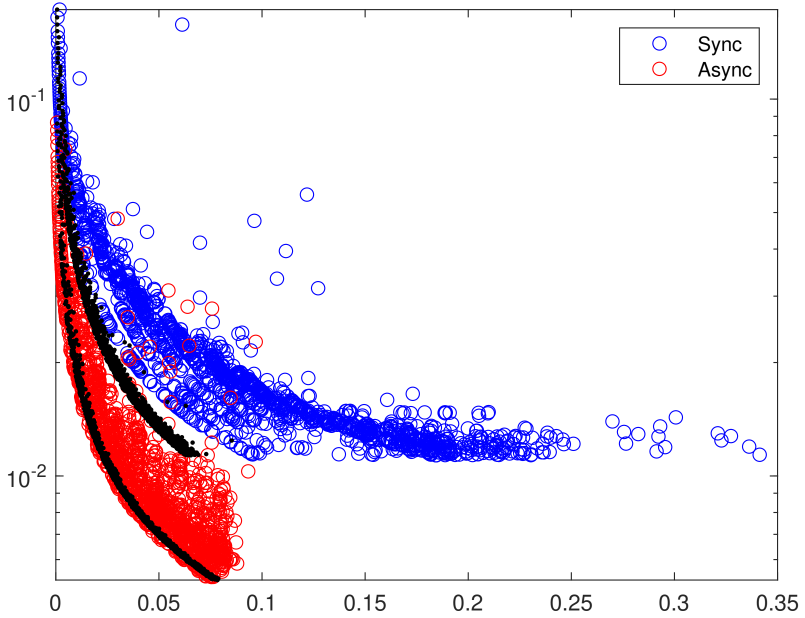
2D isotropic diffusion PDE with 5-point FD discretization, $n = 90,000$



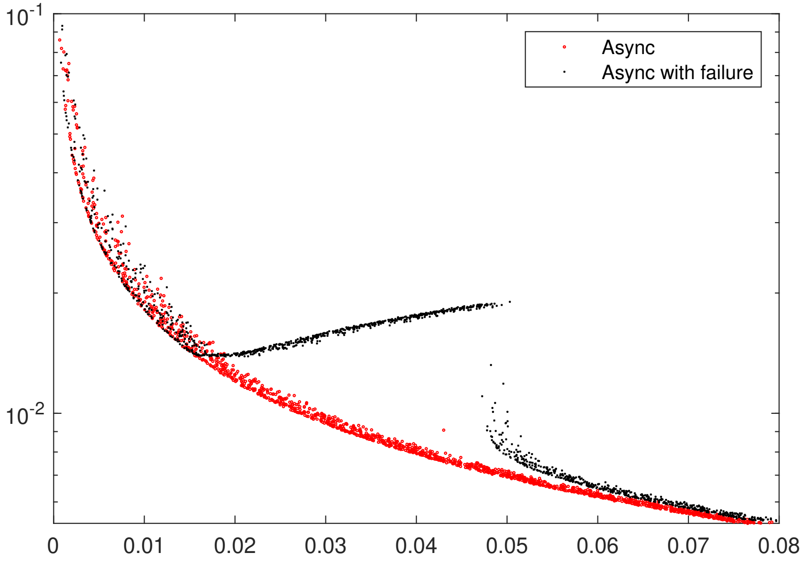
Synchronous and asynchronous Jacobi - with background processes



Synchronous and asynchronous Jacobi - with background processes



Asynchronous Jacobi - with “failure” of thread 10



Second order Richardson

To solve $Ax = b$,

$$x^{(j+1)} = x^{(j)} + \beta(x^{(j)} - x^{(j-1)}) + (1+\beta)\alpha(b - Ax^{(j)})$$

Optimal parameter values, given A is SPD and $\text{spec}(A) \subset [a, b]$:

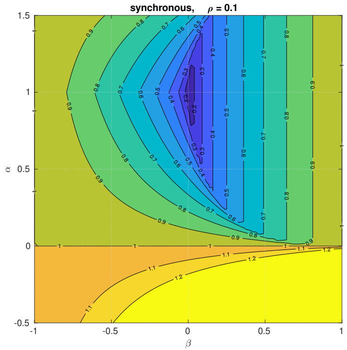
$$\alpha = \frac{2}{a+b}$$
$$\beta = \left(\frac{\sqrt{b}-\sqrt{a}}{\sqrt{b}+\sqrt{a}}\right)^2$$

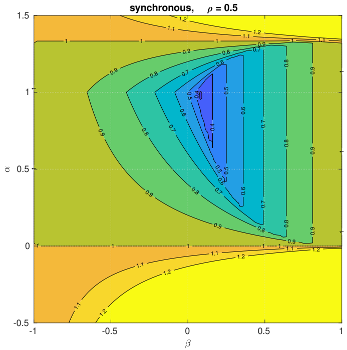
found by optimizing the spectral radius of the block 2×2 iteration matrix, $T_{\alpha,\beta}$.

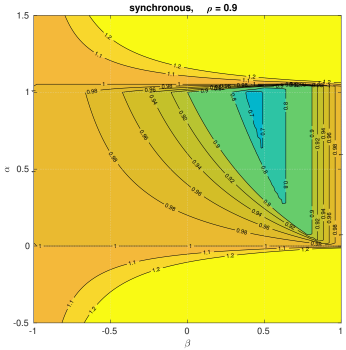
In the following figures, assume $a = 1 - \rho > 0$ and $b = 1 + \rho$.

E. Chow, A. Frommer, and D. B. Szyld, "Asynchronous Richardson iterations: Theory and practice," *Numerical Algorithms*, 2020.

Second order Richardson - Synchronous case, spectral radius of $T_{\alpha,\beta}$

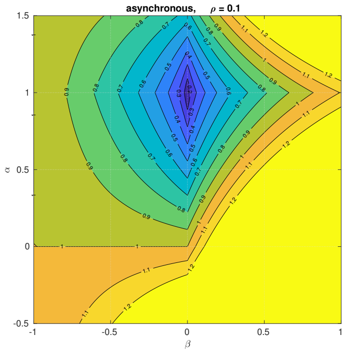


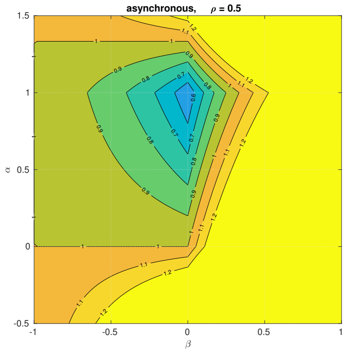


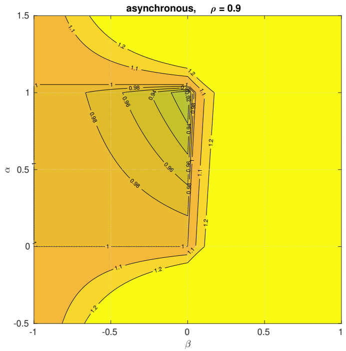


As ρ increases from 0 to 1, the optimal β increases from 0 to 1.

Second order Richardson - Asynchronous case, spectral radius of $|T_{\alpha,\beta}|$

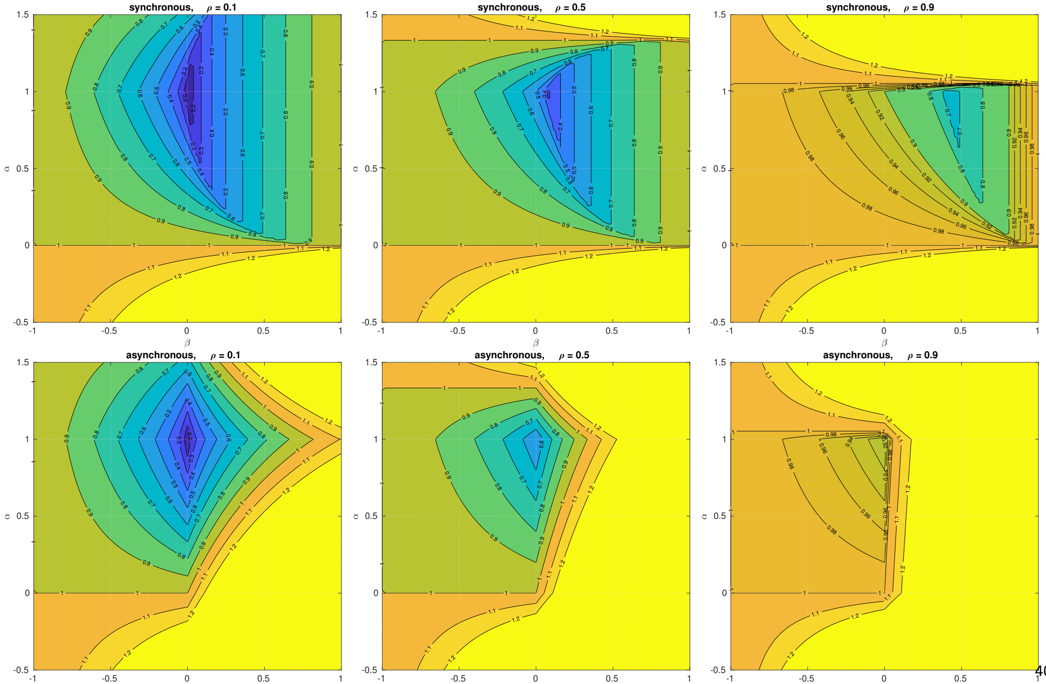






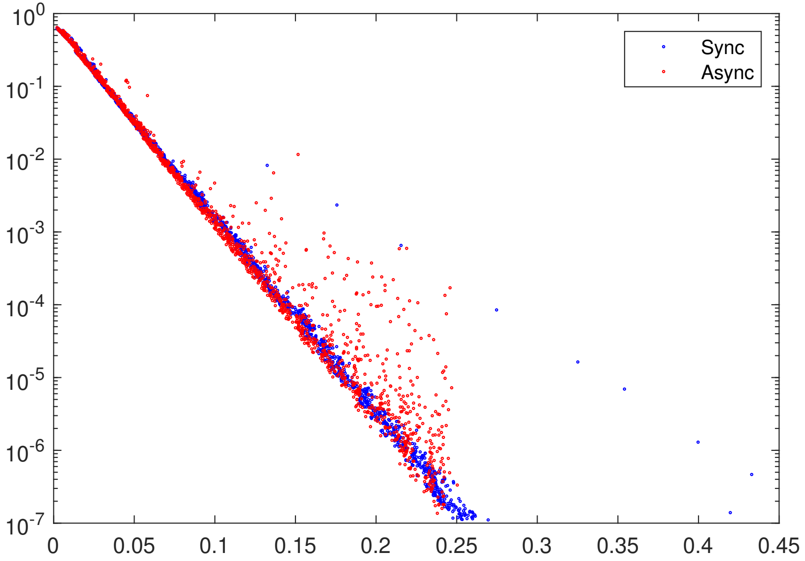
For moderate or large values of ρ, the optimal value of β gives $|T_{\alpha,\beta}| > 1$.

Second order Richardson



Second order Richardson - 5 threads

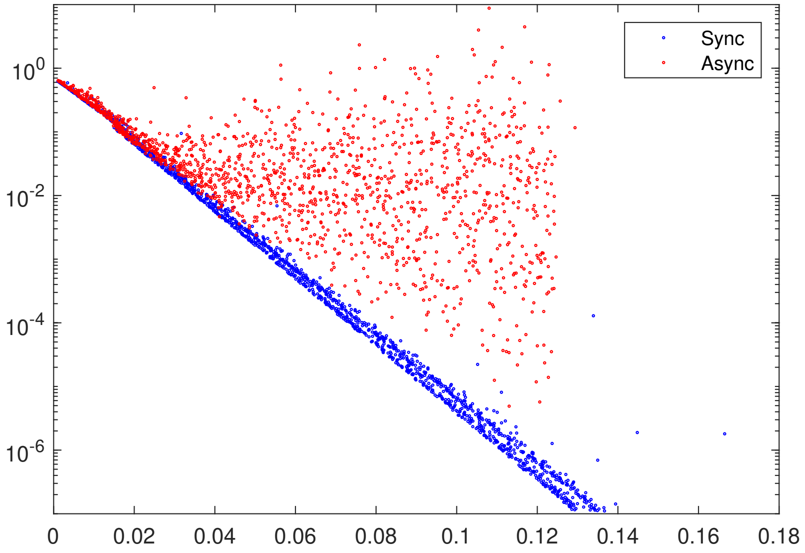
2D isotropic diffusion PDE with 5-point FD discretization, $n = 90,000$



Asynchronous convergence is possible (and likely) for 5 threads.

Second order Richardson - 10 threads

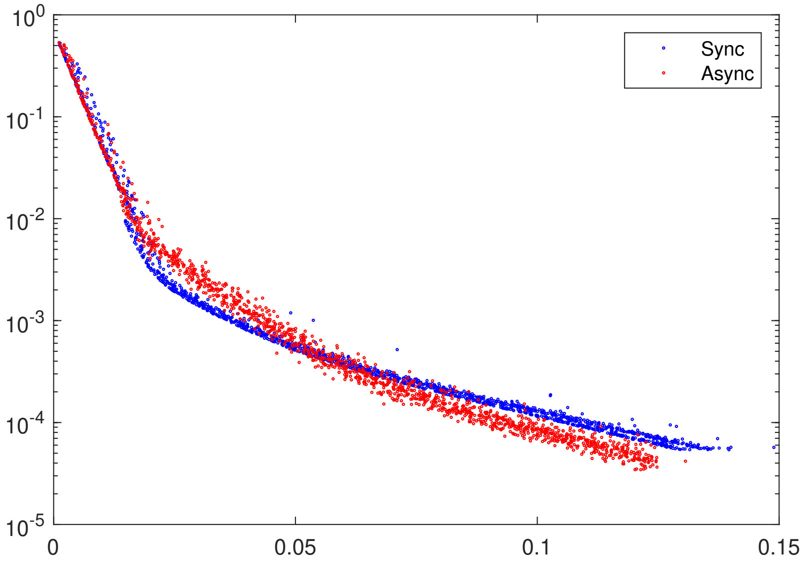
2D isotropic diffusion PDE with 5-point FD discretization, $n = 90,000$



Asynchronous convergence is degraded for 10 threads.

Second order Richardson - 10 threads - $\beta = 0.95$

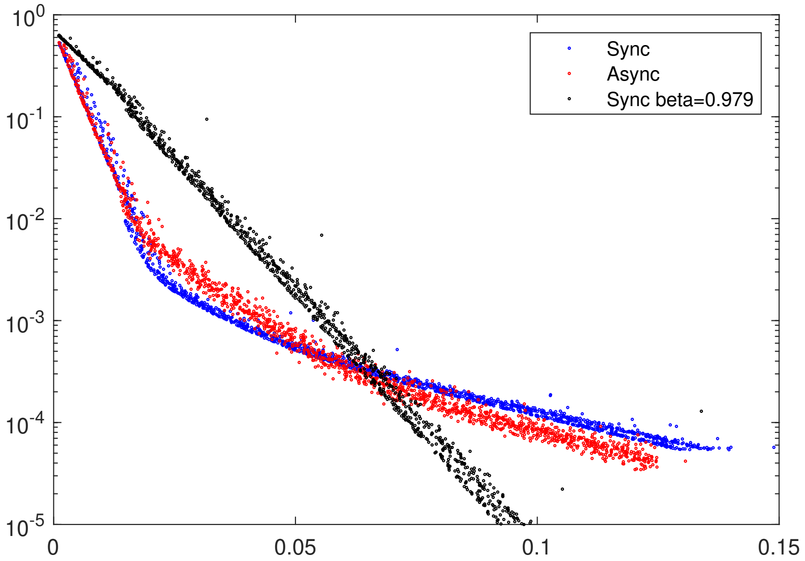
2D isotropic diffusion PDE with 5-point FD discretization, $n = 90,000$



Asynchronous convergence can be improved by using a value of $\beta = 0.95$ that is smaller than optimal $\beta = 0.979341620608331$.

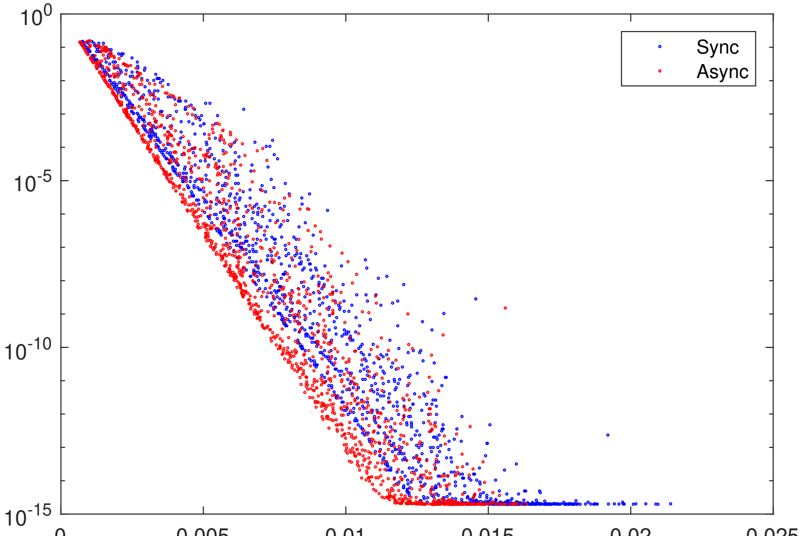
Second order Richardson - 10 threads - $\beta = 0.95$

2D isotropic diffusion PDE with 5-point FD discretization, $n = 90,000$



Asynchronous convergence can be improved by using a value of $\beta = 0.95$ that is smaller than optimal $\beta = 0.979341620608331$.

2nd order Richardson - 20 threads - shifted system (0.99)



Asynchronous iterations may converge faster than synchronous iterations for matrices that are well-conditioned. Here, $\rho = 0.9899$, $\beta = 0.752164611259891$.

References: other, advanced asynchronous solvers

Asynchronous multigrid
J. Wolfson-Pou and E. Chow, "Asynchronous multigrid methods," IPDPS, Rio de Janeiro, Brazil, May 20-24, 2019, pp. 101-110.

Asynchronous optimized Schwarz
I. Yamazaki, E. Chow, A. Bouteiller, and J. Dongarra, "Performance of asynchronous optimized Schwarz with one-sided communication," *Parallel Computing*, 86, 66-81 (2019).

Asynchronous two-level domain decomposition
C. Glusa, E. G. Boman, E. Chow, S. Rajamanickam, and D. B. Szyld, "Scalable asynchronous domain decomposition solvers," *SIAM Journal on Scientific Computing*, 2021 (to appear).

Summary

Existing theory for asynchronous iterative methods covers asymptotic convergence, but not convergence rates, which depend on properties of the parallel computation (partitioning, computer characteristics) and require a probabilistic analysis.

The numerical behavior of asynchronous iterative methods can be better than what might be expected from existing theory.

Fine-grained parallel incomplete factorization preconditioning

Conventional ILU factorization

Given sparse A, compute $LU \approx A$, where L and U are sparse.

Define S to be the sparsity pattern, $(i, j) \in S$ if l_{ij} or u_{ij} can be nonzero.

Conventional ILU factorization

Given sparse A, compute $LU \approx A$, where L and U are sparse.

Define S to be the sparsity pattern, $(i, j) \in S$ if l_{ij} or u_{ij} can be nonzero.

```
for $i = 2, \ldots, n$ do
    for $k = 1, \ldots, i-1$ and $(i, k) \in S$ do
        $a_{ik} = a_{ik}/a_{kk}$
        for $j = k+1, \ldots, n$ and $(i, j) \in S$ do
            $a_{ij} = a_{ij} - a_{ik}a_{kj}$
        end for
    end for
end for
```

Level scheduling ILU

At each step, find all rows that can be eliminated in parallel (rows that only depend on rows already eliminated).

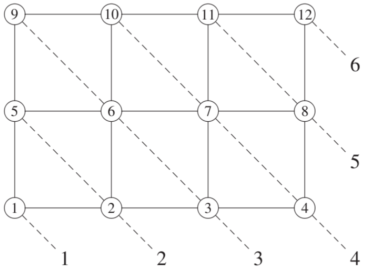


Figure from Saad 2003

Regular grids: van der Vorst 1989; Joubert & Oppe 1994
Irregular problems: Heroux, Vu, & Yang 1991; Pakzad, Lloyd, & Phillips 1997; Gonzalex, Cabaleiro, & Pena 1999; Dong & Cooperman 2011; Gibou & Min 2012; Naumov 2012
Triangular solves: Anderson & Saad 1989, Saltz 1990; Hammond & Schreiber 1992

Multicolor reordering for ILU

Multicolor reorderings can increase parallelism, but the resulting factorization is different and is a worse preconditioner.

$$\begin{bmatrix} D_1 & A_{12} & A_{13} & A_{14} \\ A_{21} & D_2 & A_{23} & A_{24} \\ A_{31} & A_{32} & D_3 & A_{34} \\ A_{41} & A_{42} & A_{43} & D_4 \end{bmatrix}$$

PCG iterations for Laplacian problem

	2D (1001^2 grid)	3D (101^3 grid)
natural	719	89
red-black	1159	142

Degradation due to ordering can be much worse for harder problems.

Refs: Poole & Ortega 1987; Elman & Agron 1989; Jones & Plassmann 1994; Nakajima 2005
GPUs: Li & Saad 2010; Heuveline, Lukarski & Weiss 2011

Domain decomposition ILU

ILU on each subdomain in parallel. Somehow eliminate the interface rows in parallel.

$$\begin{bmatrix} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & B_m & F_m \\ E_1 & E_2 & \cdots & E_m & C \end{bmatrix}$$

Convergence does not degrade if subdomains are large.
This is a coarse-grained parallelization.

Refs: Ma & Saad 1994, Karypis & Kumar 1997; Vuik et al 1998; Hysom and Pothen 1999; Magolu monga Made & van der Vorst 2002

Fine-grained parallel ILU factorization

An ILU factorization, $A \approx LU$, with sparsity pattern S has the property

$$(LU)_{ij} = a_{ij}, \quad (i,j) \in S.$$

Fine-grained parallel ILU factorization

An ILU factorization, $A \approx LU$, with sparsity pattern S has the property

$$(LU)_{ij} = a_{ij}, \quad (i,j) \in S.$$

Instead of Gaussian elimination, we compute the *unknowns*

$$\begin{array}{lll} l_{ij}, & i > j, & (i,j) \in S \\ u_{ij}, & i \le j, & (i,j) \in S \end{array}$$

using the *constraints*

$$\sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} = a_{ij}, \quad (i,j) \in S.$$

If the diagonal of L is fixed, then there are $|S|$ unknowns and $|S|$ constraints.

Solving the constraint equations

The equation corresponding to (i, j) gives

$$\begin{aligned} l_{ij} &= \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}\right), \quad i > j \\ u_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \qquad i \le j. \end{aligned}$$

The equations have the form $x = G(x)$. It is natural to try to solve these equations via a fixed-point iteration

$$x^{(k+1)} = G(x^{(k)})$$

with an initial guess $x^{(0)}$.

Admittedly a strange approach...

Write matrix factorization as a set of bilinear equations

- More bilinear equations than original equations
- Equations are nonlinear

Admittedly a strange approach...

Write matrix factorization as a set of bilinear equations

- More bilinear equations than original equations
- Equations are nonlinear

Potential advantages

- Do not need to solve the nonlinear equations exactly (no need to compute the incomplete factorization exactly)
- Nonlinear equations may have a good initial guess (e.g., time-dependent problems)
- Lots of parallelism: up to one thread per nonzero in L and U

Numerical tests

- Do the asynchronous iterations converge?
- Effect of number of threads?
- How good are the approximate factorizations as preconditioners?

Measure performance in terms of solver iteration count.

2D FEM Laplacian, $n = 203841$, RCM ordering, 240 threads on Intel Xeon Phi (KNC)

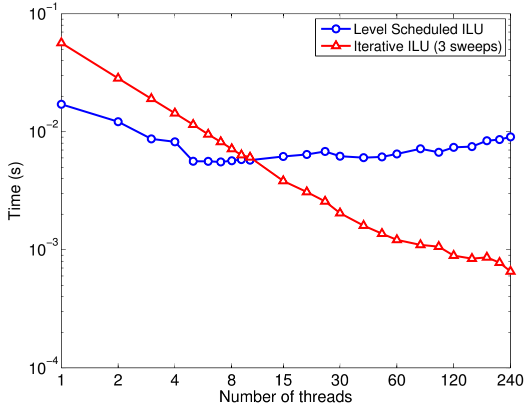
	Level 0			Level 1			Level 2		
Sweeps	PCG iter	nonlin resid	ILU resid	PCG iter	nonlin resid	ILU resid	PCG iter	nonlin resid	ILU resid
0	404	1.7e+04	41.1350	404	2.3e+04	41.1350	404	2.3e+04	41.1350
1	318	3.8e+03	32.7491	256	5.7e+03	18.7110	206	7.0e+03	17.3239
2	301	9.7e+02	32.1707	207	8.6e+02	12.4703	158	1.5e+03	6.7618
3	298	1.7e+02	32.1117	193	1.8e+02	12.3845	132	4.8e+02	5.8985
4	297	2.8e+01	32.1524	187	4.6e+01	12.4139	127	1.6e+02	5.8555
5	297	4.4e+00	32.1613	186	1.4e+01	12.4230	126	6.5e+01	5.8706
IC	297	0	32.1629	185	0	12.4272	126	0	5.8894

Very small number of sweeps required

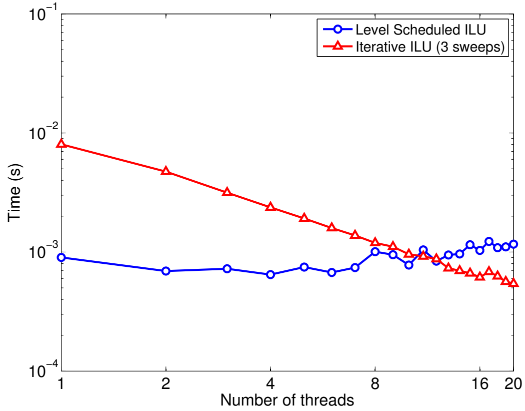
E. Chow and A. Patel, Fine-grained Parallel Incomplete LU Factorization, SIAM Journal on Scientific Computing, 37, C169-C193 (2015).

H. Anzt, E. Chow, and J. Dongarra, ParILUT - A New Parallel Threshold ILU Factorization, SIAM Journal on Scientific Computing, 40, C503-C519 (2018).

Timing comparison, ILU(2) on 100×100 grid (5-point stencil)



Intel Xeon Phi



Intel Xeon E5-2680v2, 20 cores

Results for NVIDIA Tesla K40c

SPD matrices from SuiteSparse Collection

	PCG iteration counts for given number of sweeps							Timings [ms]		
	0	1	2	3	4	5	IC	5 swps	IC	s/up
apache2	1430	1363	1038	965	960	958	958	8.8	61.	6.9
ecology2	2014	1765	1719	1708	1707	1706	1705	6.7	107.	16.0
G3_circuit	1254	961	968	993	997	997	997	12.1	110.	9.1
offshore	428	556	373	396	357	332	330	25.1	219.	8.7
parabolic_fem	763	636	541	494	454	435	393	6.1	131.	21.6
thermal2	1913	1613	1483	1341	1411	1403	1398	15.7	454.	28.9
2D Lap	653	703	664	621	554	551	550	7.4	112.	15.2
3D Lap	43	37	35	35	35	35	35	47.5	94.	2.0

IC denotes the exact factorization computed using the NVIDIA cuSPARSE library.

Iterative and approximate triangular solves

Trade accuracy for parallelism
Approximately solve the triangular system $Rx = b$

$$x^{(j+1)} = (I - D^{-1}R)x^{(j)} + D^{-1}b$$

where D is the diagonal part of R.

- implementations depend on SpMV
- iteration matrix $T = I - D^{-1}R$ is strictly triangular and has spectral radius 0 (trivial asymptotic convergence)
- for fast convergence, want the norm of T to be small
- R from stable ILU factorizations of physical problems are often close to being diagonally dominant

E. Chow, H. Anzt, J. Scott, and J. Dongarra, Using Jacobi Iterations and Blocking for Solving Sparse Triangular Systems in Incomplete Factorization Preconditioning, Journal of Parallel and Distributed Computing, 119, 219-230 (2018).

Summary

- Approach: writing matrix factorizations as a set of nonlinear equations
- Efficient methods for parallel triangular solves are necessary